CONTENTS

1 Quickstart .................................................. 3
  1.1 Why SpiX? .............................................. 3
  1.2 The \texttt{.tex} file .................................. 3
  1.3 Compilation ........................................... 4

2 Support ..................................................... 5
  2.1 License ................................................ 5
  2.2 Help! .................................................. 5

3 Why SpiX? .................................................. 7
  3.1 Which problems does SpiX solve? ...................... 7
  3.2 Why not using any other tool? ......................... 8
  3.3 Why is it named SpiX? ................................. 9

4 Download and Install ...................................... 11

5 Usage ....................................................... 13
  5.1 Configuration ......................................... 13
  5.2 Allowed commands .................................... 14
  5.3 Environment variables ................................ 15
  5.4 About errors .......................................... 15
  5.5 Command line arguments ............................... 15
  5.6 Warning ................................................ 16

6 Frequently asked questions ............................... 17
  6.1 Why won’t SpiX accept any option to control compilation? 17
  6.2 How to re-run SpiX as soon as a file has changed? .......... 17
  6.3 How to run SpiX on several files? ...................... 18
  6.4 How to check if a \texttt{.tex} file has any SpiX commands set? 18
  6.5 How to stop compilation on first error? ................. 19
  6.6 How to go on compiling, even when errors occur? ....... 20
SpiX is yet another compilation tool\footnote{https://www.ctan.org/topic/compilation} for .\texttt{tex} files. It aims at being simple and human readable. Every piece of configuration is written in the .\texttt{tex} file itself, in a clear format (a list of console commands).

\textit{Quickstart} (page 3) should give you enough information to start using SpiX. License and links are given in \textit{Support} (page 5). In \textit{Why SpiX?} (page 7), you can find out if you should use SpiX, or if you should prefer another tool. To install SpiX, read \textit{Download and Install} (page 11). The detailed SpiX manual is in \textit{Usage} (page 13). At last, \textit{Frequently asked questions} (page 17) give answers to some questions you might have.
1.1 Why SpiX?

With SpiX, the compilation process of a .tex file (Is it compiled using latex? pdflatex? xelatex? lualatex? Should I process its bibliography? with bibtex or biber? Is there an index?) is written in the .tex file itself, in a human-readable format (a shell script). That way:

- when you want to compile two years later, you don’t have to guess the compilation process;
- you can send the .tex file to someone, and that’s it: no need to send detailed instructions or a Makefile along with it (everything is in the .tex file);
- the compilation process is human readable: it can be understood by anyone who is able to read a very basic shell script. In particular, one can read it even if she does not know SpiX.

1.2 The .tex file

Write the compilation process of your .tex file as a shell script, before the preamble, as lines starting with \$:

```
\$ Compile this file twice with lualatex.
\$ lualatex foo.tex
\$ lualatex foo.tex

\documentclass{article}
\begin{document}
Hello, world!
\end{document}
```

You can also replace the file name with $\texttt{texname}$ (and $\texttt{basename}$, without the extension). That way, you don’t have to worry about the file name when writing your commands.

```
\$ Compile this file twice with lualatex.
\$ lualatex $\texttt{texname}$
\$ lualatex $\texttt{texname}$
```

\footnote{A more detailed answer to “Why SpiX?” can be found in *Why SpiX?* (page 7).}
1.3 Compilation

To compile the .tex file, run SpiX:

```bash
spix foo.tex
```

Spix will parse the .tex file, looking for shell snippets (lines before the preamble starting with `%%`), and run them.

That’s all!
2.1 License

SpiX is licensed under the Gnu GPL 3 license\(^3\), or any later version.

2.2 Help!

- The home page of SpiX is: http://framagit.org/spalax/spix.
- Documentation is at: http://spix.rtfd.io.
- To report bugs, or ask for help, visit: https://framagit.org/spalax/spix/issues (if you don’t feel like creating yet another account, you can send me an email at spalax(plus)python(at)gresille(dot)org).

\(^3\) https://www.gnu.org/licenses/gpl-3.0.html
3.1 Which problems does SpiX solve?

The goal of SpiX is to have every information about a .tex compilation process inside the very file to process.

3.1.1 Example 1

Alice is a math teacher. She writes every document she shows or hands out to her students using LaTeX. She has a repository consisting of hundreds of LaTeX files. Most of her documents are compiled using a single pass of LuaLaTeX, but some of them need two passes (because labels and references), some of them contains pstricks figures that must be compiled with LaTeX, then converted to PDF (because she copied them from another repository that uses LaTeX)…

When she works on a file she edited one year ago, with her previous class, she has to guess how to compile it (lualatex? lualatex+lualatex? latex+dvipdf?).

Using SpiX, the compilation process is written inside the .tex file, so she can:

- look at it to see which tool to use to compile it;
- compile it using SpiX.

3.1.2 Example 2

Alice happens to works with Bob, who also uses LaTeX. The ideal way to work on the same file would be to share a git repository containing a Makefile, but evoking those tools would scare Bob away. So they exchange files via email. Using SpiX, the compilation process of the file they exchange is written inside the file itself:

```
$ Use lualatex twice to compile this file:
$% $ lualatex foo.tex
```

(continues on next page)

---

4 https://framagit.org/lpaternault/cours-2-math
5 https://tug.org/PSTricks/
6 https://www.apmep.fr/-Annales-Bac-Brevet-BTS-
%$ lualatex foo.tex

\documentclass{article}
\begin{document}
Hello, world!
\end{document}

- Alice: The first three lines of this file can be parsed by SpiX, so that Alice simply runs spix foo to compile it;
- Bob: The first three lines of this file are human-readable, so Bob understands how he should compile it.

3.2 Why not using any other tool?

3.2.1 Makefile

If your project is complex (convert images, compile .dot graphs, several latex passes, bibliography, index...), use a Makefile. You may prefer SpiX if:

- the Makefile would be only two lines long;
- you have tens or hundreds of simple .tex files, with slightly different compilation processes (which would mean tens or hundreds of Makefiles, or a huge Makefile);
- you want to have the compilation process inside the .tex file itself.

3.2.2 Arara

I got the idea to write compilation information into the .tex file itself from Arara\textsuperscript{7}.

Arara provides a set of rules to compile files. If something is missing, you can write your own rule in an external file, so you might prefer SpiX if you want everything in the same .tex file.

Arara configuration is written using YAML. So, to understand Arara configuration, one has to know YAML and Arara (while SpiX configuration is plain shell commands, so it is human readable\textsuperscript{17}).

You might prefer Arara if you have complex rules; SpiX is well suited for plain, simple commands.

\textsuperscript{7} https://gitlab.com/islandoftex/arara
\textsuperscript{17} At least, readable by anyone who can use a terminal.
3.2.3 TrY

TrY\(^8\) does exactly what SpiX does (and I copied the syntax of commands in \texttt{.tex} files from TrY). But it is written in Python2 (which is obsolete\(^9\)), and it seems to be no longer maintained\(^10\). SpiX can be seen as a successor of TrY\(^18\).

3.2.4 Latexmk

Latexmk\(^11\) (and similar tools\(^12\)) has a slightly different purpose.

- It guesses how to compile file (how many passes, etc.), while SpiX commands are explicit (there is no magic in SpiX).
- User has to specify which flavor (LaTeX, pdflatex, LuaTeX, XeLaTeX…) to use, while with SpiX, this is stored in the \texttt{.tex} file.

3.3 Why is it named SpiX?

Arara\(^13\) is named after the blue-and-yellow macaw\(^14\) (\textit{arara} meaning \textit{macaw} in Portuguese), which is a big parrot. This project, which aims at being a simple version of Arara, is named after the blue winged parrotlet\(^15\) (\textit{toui de Spix} in French), which is a small parrot.

Obviously, the capital \textit{X} is a nod to the capital \textit{X} of LaTeX.

---

\(^8\) https://ctan.org/pkg/try
\(^10\) https://bitbucket.org/ajabutex/try/issues/14/is-this-project-still-maintained
\(^11\) http://personal.psu.edu/jcc8/latexmk/
\(^12\) https://www.ctan.org/topic/compilation
\(^13\) https://gitlab.com/islandoftex/arara
\(^14\) https://en.wikipedia.org/wiki/Blue-and-yellow_macaw
\(^15\) https://en.wikipedia.org/wiki/Blue-winged_parrotlet
\(^16\) https://commons.wikimedia.org/w/index.php?curid=79073013
Fig. 1: A blue winged parrotlet. Photo by Evaldo Resende - Own work, CC BY-SA 4.0\textsuperscript{16} (flipped, resized, reframed by Louis Paternault).
• The preferred installation method uses pip\(^{19}\) (preferably in a virtualenv\(^{20}\)):

```bash
python3 -m pip install spix
```

• You can install it from sources:
  - download the stable\(^{21}\) or development\(^{22}\) version;
  - unpack it;
  - install it (in a virtualenv\(^{23}\), if you do not want to mess with your distribution installation system):

```bash
python3 -m pip install .
```

• To install it from CTAN\(^{24}\):
  - download\(^{25}\) the package from CTAN;
  - extract the spix.py file, and copy it somewhere in your PATH. On GNU/Linux (and MacOS?), you can rename it to spix.

• Quick and dirty Debian (and Ubuntu?) package

  This requires stdeb\(^{26}\) to be installed:

```bash
python3 setup.py --command-packages=stdeb.command bdist_deb
sudo dpkg -i deb_dist/spix-<VERSION>_all.deb
```

---

\(^{19}\) [https://pypa.io](https://pypa.io)

\(^{20}\) [https://docs.python-guide.org/dev/virtualenvs/](https://docs.python-guide.org/dev/virtualenvs/)

\(^{21}\) [https://pypi.python.org/pypi/spix](https://pypi.python.org/pypi/spix)

\(^{22}\) [https://framagit.org/spalax/spix-/archive/master/spix-master.zip](https://framagit.org/spalax/spix-/archive/master/spix-master.zip)

\(^{23}\) [https://docs.python-guide.org/dev/virtualenvs/](https://docs.python-guide.org/dev/virtualenvs/)

\(^{24}\) [https://ctan.org/](https://ctan.org/)

\(^{25}\) [https://ctan.org/pkg/spix](https://ctan.org/pkg/spix)

\(^{26}\) [https://github.com/astraw/stdeb](https://github.com/astraw/stdeb)
5.1 Configuration

To configure how your .tex file is compiled, simply write the necessary commands before your preamble, preceded with \%. That's all:

```latex
% Compile this file using latex+dvipdf:
%\
%% latex foo.tex
%% dvipdf foo.dvi

\documentclass{article}
\begin{document}
Hello, world!
\end{document}
```

Now, when calling SpiX on this file, commands `latex foo.tex` and `dvipdf foo.dvi` are called:

```
spix foo.tex
```

Note:
• The lines that are interpreted as snippets by SpiX must begin exactly with the two characters
\%\$ followed by a space. Any other prefix is not considered a command:

\%
A command
\%
Ignored
\%
Ignored
\%
Ignored
\%
Ignored

• Any snippet defined after the beginning of the preamble is ignored. SpiX does not parse
LaTeX code, so it considers any line that is not empty, or does not begin with % (maybe
preceded by spaces) as a preamble.

\%
A snippet
% \documentclass\{article\}
\%
Ignored
% \begin\{document\}
\%
Ignored
% \end\{document\}
\%
Ignored

Note: There is no configuration file. SpiX is meant to run the same way on any machine: you
set up configuration in a .tex file, you send this file to your friend, she runs SpiX on it, and it
runs exactly the same way (not relying on a configuration file located somewhere in your home
directory, that you forgot to send along the .tex file).

5.2 Allowed commands

The code snippets defined in SpiX are interpreted by the sh shell\footnote{Which default to dash on Debian, for instance.} (but try to stick to valid sh
code, to make your snippet portable). This means that variables and control structures are allowed.

\%
dviname=$basename.dvi
\%
ltx $texname
\%
bibtex $basename
\%
for i in $(seq 3)
\%
do
\%
ltx $(seq 3)
\%
endfor
\%
dvipdf $dviname

Consecutive lines starting with %\$ are interpreted by one single shell call.
$myvariable=foo
#$ This would display "foo"
#$ echo $myvariable
% This line does not start with "$\$", starting another shell.
#$ # This would display nothing, since "$myvariable" has been
→ defined in another shell.
#$ echo $myvariable

5.3 Environment variables

In order to be readable by a person who has never heard about SpiX, the snippets are run as-is
(interpreted by the sh shell).

A few environment variables are introduced (this allows snippets to be independent on file name).
For instance, suppose Donald is writing his next book, in ~/taocp/vol7.tex:

• $texname is the file name (without directory): vol7.tex;
• $basename is the file name, without extension: vol7.

For instance, if file foo.tex contains the following snippet:

%$ latex $texname
%$ dvipdf $basename

When calling SpiX, commands latex foo.tex and dvipdf foo are run.

5.4 About errors

SpiX will stop compilation when a code snippets fails (returns an error code different from 0).
To change this behavior, see How to stop compilation on first error? (page 19) or How to go on
compiling, even when errors occur? (page 20).

5.5 Command line arguments

Since there is no option to configure how compilation is performed (everything is in the .tex
file), the binary has very few options.

Compile a .tex file, executing commands that are set inside the file itself.

5.5.1 Positional Arguments

**FILE**  
File to process.

5.5.2 Named Arguments

- **-n, --dry-run**  
  Print the commands that would be executed, but do not execute them.  
  Default: False

- **--version**  
  Show version and exit.

5.6 Warning

SpiX is dumb: it does not control what is run, it does not check that it is safe to run. It runs what it is told to run. For instance:

- it does not prevent malicious commands:
  
  ```
  %$ rm -fr /
  ```

- it does not prevent infinite loops:
  
  ```
  %$ spix $texname
  ```

- it does not prevent fork bombs:
  
  ```
  %$ spix $texname & spix $texname &
  ```

Basically, calling SpiX is like running a shell script: do not call SpiX on an untrusted .tex file.
FREQUENTLY ASKED QUESTIONS

- Why won’t SpiX accept any option to control compilation? (page 17)
- How to re-run SpiX as soon as a file has changed? (page 17)
- How to run SpiX on several files? (page 18)
- How to check if a .tex file has any SpiX commands set? (page 18)
- How to stop compilation on first error? (page 19)
- How to go on compiling, even when errors occur? (page 20)

6.1 Why won’t SpiX accept any option to control compilation?

The purpose of SpiX is to have every single piece of information regarding how to compile a .tex file inside the .tex file itself. So, SpiX having options to control the compilation would go against this purpose. That is why the only SpiX options are options about SpiX itself (--help, --version), or about the compilation (--dry-run), but nothing that changes how the file is to be compiled.

6.2 How to re-run SpiX as soon as a file has changed?

SpiX has no built-in feature to do this, but you can use external tools, like entr28:

```
ls foo.tex | entr spix foo.tex
```

But if your compilation process includes several passes of LaTeX, and biblatex, and…, you probably don’t want to re-run everything as soon as you fix a typo in your document. In this case, do not use SpiX at all:

---

28 https://eradman.com/entrproject/
 SpiX, Release 1.1.0

Then, every single type has been fixed, at last, you can use SpiX to properly compile your document:

```
spix foo.tex
```

### 6.3 How to run SpiX on several files?

SpiX accepts exactly one file as an argument.
To run it on several files, you can use `find`:

```
find . -name '*tex' -exec spix {} \;
```

or `parallel`:

```
parallel spix -- *tex
```

or both:

```
find . -name '*tex' -exec parallel spix -- {} \+
```

### 6.4 How to check if a .tex file has any SpiX commands set?

Option `--dry-run` will print the code snippets to be run by SpiX. Thus, to test whether any code snippets has been set in a .tex file, you can test use:

```
if [ -z "$(spix --dry-run foo.tex)" ]
then
  echo "No command defined."
else
  echo "Some commands defined."
fi
```

---

29 [https://www.gnu.org/software/findutils/](https://www.gnu.org/software/findutils/)

30 [https://www.gnu.org/software/parallel/](https://www.gnu.org/software/parallel/)
6.5 How to stop compilation on first error?

A code snippet defined in your .tex file is executed, even if commands inside it fails. For instance, suppose your file contains the following code snippet.

```
\$ latex $texname
\$ bibtex $basename
\$ latex $texname
\$ latex $texname
\$ dvipdf $basename
```

If the first LaTeX compilation fails, the following commands are still executed. Preventing any further command to be executed is dealt with using the shell options, not SpiX. You can chain your commands using `&&`:

```
\$ latex $texname &&
\$ bibtex $basename &&
\$ latex $texname &&
\$ latex $texname &&
\$ dvipdf $basename
```

or use `set -e`:

```
\$ set -e
\$ latex $texname
\$ bibtex $basename
\$ latex $texname
\$ latex $texname
\$ dvipdf $basename
```

or split you snippet into several snippets (that way, SpiX will stop after the first code snippet that ends with an error):

```
\$ latex $texname
%
\$ bibtex $basename
%
\$ latex $texname
%
\$ latex $texname
%
\$ dvipdf $basename
```
6.6 How to go on compiling, even when errors occur?

If any code snippets ends with an error (an error code other than 0), SpiX will stop the compilation. You may want to continue, no matter what. Once again, this is achieved using the shell, not using SpiX. You can:

- force the error code at the end of your code snippet:

  ```bash
  $ latex $texname
  $ exit 0
  ```

- Catch errors using `|| true`:

  ```bash
  $ latex $texname || true
  ```